**编码引论第一次大作业**

**分组码与网格编码**

**实验报告**

姓 名：张鹤龄

班 级：无04

学 号：2020010735

完成时间：2022/10/20

目录

[一、实验分工 3](#_Toc117355805)

[二、设计思路与过程 3](#_Toc117355806)

[一、线性码 3](#_Toc117355807)

[二、卷积码 4](#_Toc117355808)

[编码模块 4](#_Toc117355809)

[译码模块 6](#_Toc117355810)

[三、CRC校验 10](#_Toc117355811)

[四、信道设计 11](#_Toc117355812)

[信道建模场景 11](#_Toc117355813)

[映射方式 12](#_Toc117355814)

[三个场景的硬判决解映射讨论 12](#_Toc117355815)

[三个条件的讨论 13](#_Toc117355816)

[三、仿真结果及分析 14](#_Toc117355817)

[信道部分 14](#_Toc117355818)

[星座图 14](#_Toc117355819)

[内核与外部信噪比关系 15](#_Toc117355820)

[硬判决误比特率与信噪比关系 15](#_Toc117355821)

[编译码部分 19](#_Toc117355822)

[线性码 19](#_Toc117355823)

[卷积码 20](#_Toc117355824)

[四、特定传输任务中编译码及映射方式的选择 24](#_Toc117355825)

[五、实验心得 25](#_Toc117355826)

[附录 主要文件清单及说明 26](#_Toc117355827)

# 一、实验分工

本实验的小组分工如下：

张鹤龄：线性码编译码、CRC校验、Viterbi似然判决推导

唐钰凯：符号电平映射、信道模块、解映射

邵晨扬：1/2、1/3效率卷积码模块，Viterbi判决译码器（硬/软\*）

# 二、设计思路与过程[[1]](#footnote-1)

## 一、线性码

线性码设计的基本思路为[[2]](#footnote-2)：

* 确定原码长度k和许用码字长度n；
* 寻找k个长度为n且线性独立的码作为基向量，用这些基向量线性组合出所有的可行码字；
* 将原码右乘基向量矩阵即可编码；
* 解码时，需要找到错误图案。为此需要设计校验矩阵，使接收码字中非错误图案部分消去，而错误图案留下。
* 代数原理保证一定存在一个(n-k)×n校验矩阵H，使得H的零空间恰好是许用码字集，这样通过将接收码字右乘H，即可分离出错误图案e对应的校正子eH。实验中根据课件设定H。
* 将接收的码字右乘校验矩阵，得到校验子。校验子总对应一个或一个以上的错误图案，再由最大似然准则，选择最轻的错误图案并将其从接收到的码字中减去，得到一个许用码字。用这个许用码字即可复原原码，完成解码过程。

**编码过程的核心代码如下：**

1. bitstream(8\*(ii-1)+1:8\*ii) = mod(codepatch\*G,2);

其中，codepatch为从原码流中裁剪下的3bit片段，通过线性编码（右乘编码矩阵G）得到编码后的许用码字，并放入输出比特流bitstream中。

**解码过程的核心代码如下：**

一、准备校正子表

1. for ii = 1:5
2. C1 = nchoosek(n,ii);
3. [lengthC,~] = size(C1);
4. for jj = 1:lengthC
5. temp = zeros(1,8);
6. temp(C1(jj,:)) = 1;
7. errorset(pointer,:) = temp;
8. pointer = pointer+1;
9. end
10. end
11. corset = mod(errorset\*H',2);

以上代码的基本思想是，生成所有可能的错误图案序列，并将其和校验矩阵运算得到校正子—错误图案键值对。由于校验矩阵的秩是5，在最大似然判断下，错误图案的码重也不可能超过5，因此这里只生成码重不超过5的错误图案及对应校正子。

考虑到校正子只有32个，而以上的键值对有219个，显然浪费了空间和时间。在这些键值对中，每个校正子可能对应不止一个错误图案，而根据最大似然原则应该选择最轻的一个作为与其对应的错误图案。代码如下：

1. for jj = 1:length(corset)
2. if corset(jj,:) == s
3. temp = sum(errorset(jj,:));%计算错误图案码重
4. if temp <= Min
5. Min = temp;
6. index = jj;
7. end
8. end
9. end
10. new\_errorset(ii,:) = errorset(index,:);

所有32个校正子按照二进制表示，可以组成一株完全二叉树，二叉树的每个节点中存储对应的错误图案。由于完全二叉树可以和整数序列一一对应，我们可以用一次查找就找到校正子对应的错误图案，再将其从原码中减去得到许用码字。代码如下：

1. bitpatch = bitstream(8\*(ii-1)+1:8\*ii);%获得接收码字
2. corrector = mod(bitpatch\*H',2);%计算校正子
3. index = sum(corrector.\*[16,8,4,2,1])+1;
4. er = errorset(index,:);%找到对应的错误图案
5. originpatch = mod(bitpatch-er,2);%恢复许用码字

之后再根据许用码字得到原码，译码工作即告完成。

## 二、卷积码

### 编码模块

本实验用到的编码模块为Convol.code.m。各参数含义如下：

**输入**

infoSeq: information sequence 需要编码的01信息序列

mode: method to coding: --1:(15,17) --2:(13,15,17) 选择编码的方式

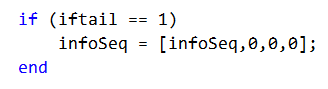
iftail: --0: notail --1:tail 选择是否收尾

**输出**

输出为编码后的码序列。

本实验采用收尾方式的卷积码编码。具体来说，为保证卷积码最终只有一个判决路径，在编码时可以主动在需编码的信息序列最后缀上m个0（其中m为寄存器个数）以使编码器最终停在全0的确定状态上。这样，在译码时就可以确定编码器的最终状态（全0），并从这个状态回溯判决路径以得到译码结果。

为实现收尾编码首先在infoseq后缀3个0：



之后按照推荐的卷积参数构造如图所示的编码器：

**1/2效率**

图示

中度可信度描述已自动生成

**1/3效率**

图示, 示意图

描述已自动生成

**编码器的核心代码见下：**

1. if (mode == 1)
2. d = zeros(1,3); % Temporary state
3. for i = 1:len
4. m = dot([1,1,0,1],[infoSeq(i),d]);
5. n = dot([1,1,1,1],[infoSeq(i),d]);
6. y1 = (mod(m,2)==1) | (m==1);
7. y2 = (mod(n,2)==1) | (n==1);
8. d = [infoSeq(i),d(1:2)];
9. tmpout = [tmpout,y1,y2];
10. end
11. else
12. d = zeros(1,3); % Temporary state
13. for i = 1:len
14. m = dot([1,0,1,1],[infoSeq(i),d]);
15. n = dot([1,1,0,1],[infoSeq(i),d]);
16. p = dot([1,1,1,1],[infoSeq(i),d]);
17. y1 = (mod(m,2)==1) | (m==1);
18. y2 = (mod(n,2)==1) | (n==1);
19. y3 = (mod(p,2)==1) | (p==1);
20. d = [infoSeq(i),d(1:2)];
21. tmpout = [tmpout,y1,y2,y3];
22. end
23. end
24. out = tmpout;

代码中，在每一个编码帧，寄存器中存储的序列和推荐的参数做内积，将结果输出到编码序列中。之后，每个寄存器读入自身的输入。

### 译码模块

卷积码的译码使用Viterbi算法。Viterbi算法的基本思想为基于编码器状态转移的动态规划，基本步骤如下：

1、第0时刻，编码器处于全零状态。除了零状态初始度量设置为0外，其他状态均设置初始度量为无穷；

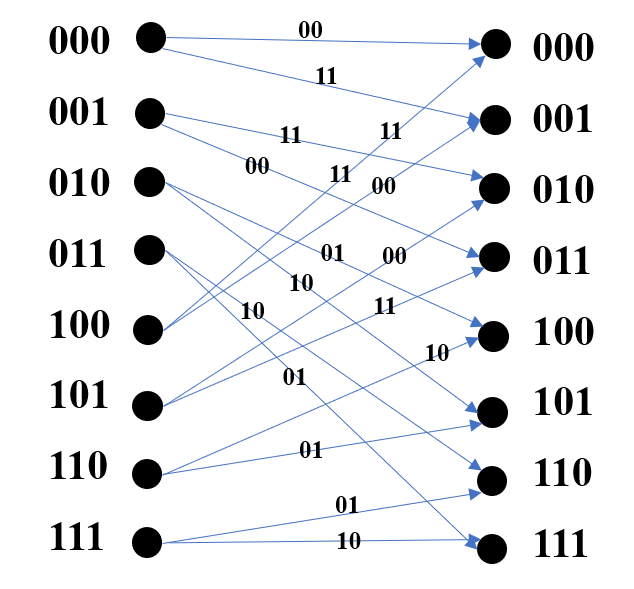
2、Viterbi算法的目标是使得度量最大，从而找到最可能的译码。从而，在任一时刻，对每一个状态只记录到达路径中度量最大的一个，并计算状态度量=前一状态度量+路径度量；

3、重复以上步骤，直到编码流的结尾。对于收尾Viterbi译码，从零状态开始回溯，得出译码结果。

此外，在Viterbi译码过程中还有一些需要注意的要点。以下是负责本部分编写的同学的心得：

1. ***需要维护哪些信息？****每个时刻的每个状态无疑都需要存储回溯点（状态值）信息。那最优值呢（最大度量），需要存储每个时刻的每个状态的到达最大度量嘛？其实不用，因为t时刻的最优度量值，仅仅对t+1时刻的最优值计算有用，对t+1后面时刻的度量值计算没有任何价值。所以只需要维护一个时刻的全部状态的最优度量就可以了，然后在时间推进过程中动态更新。这样可以节省不少内存。综上，对于本次实验的参数3个寄存器，共有8个状态，所以需要建立的回溯点矩阵size为：8×length(info)，而最优度量值只用一个8×1的矩阵就可以了。*
2. ***对于当前的一个状态，如何计算两条到达路径的前置状态及输入符号？****其实，前置状态和输入符号是完全可以通过当前时刻的状态推导出来的。*

*比如当前状态是000。那么前置路径确定为两条：状态000->输入0 输出码值00；以及状态100->输入0 输出码值11。这个过程可以用寄存器图的思路轻松推导出来。这也就意味着所有的路径信息不需要存储，而通过当前的状态信息就可以反推出到达这个状态的路径值以及前置状态。其实完全可以事先计算好所有的情况（如下图所示），然后把路径信息给存储起来，但是为了程序的扩展性和可复用性，我还是用代码来实现从状态反推路径。*



总结起来，就是：

1、因为动态规划问题的无后效性，我们并不需要存储每一时刻的度量，而只需要存储当前译码帧前一帧的度量；

2、通过当前的状态信息可以反推出到达这个状态的路径值以及前置状态，因此网格拓扑不需要存储。

#### 软判决度量讨论

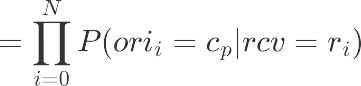
如上所述，Viterbi判决的核心在于计算每一步的度量，而如何确定度量的表达式最为关键。本次实验中，我们设计了三种软判决度量：

* 基于最大后验的软判决
* 基于最大似然的软判决
* 基于距离的软判决

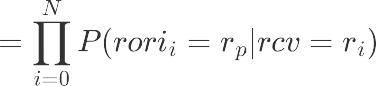
卷积码译码的过程，本质上是基于接收到的信号流判断最可能原码的过程。用条件概率表达为，



上式中r为全部的接收复电平流，c为某种原码序列cp1cp2……cpn。每个cp的取值和映射bit数有关，例如若采取2bit映射，则cp可取的值为0x0,0x1,0x2,0x3。假设信道是无记忆的，则上式可重写为



其中orii和ri分别为原码的第i位和第i个译码帧接收到的复电平。由于每位原码取值和卷积码在第i个编码帧编码出的值一一对应，上式又可重写为



其中rorii为卷积码在第i个编码帧编码出的值映射到的复电平，rp为映射时的标准复电平，若选择将m个bit映射到同一复电平上，则rp共有2m个。而我们所要做的就是使得等式右边的表达式取得最大值。至此，我们将求解最可能原码的问题转化成了：在每一译码帧，基于信道出口接收的复电平，找到其在信道入口最可能的取值。

**基于最大后验的软判决**

对于帧序号i，通过贝叶斯公式可以导出：





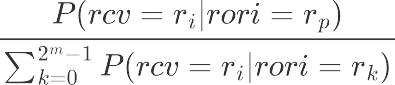
再由全概率公式：





其中m为一个复电平对应的比特数量。

假设rorii取每种复电平的概率都是相等的，则上式变为

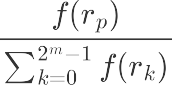


这样我们只需要计算分子上的概率。对于加性高斯噪声信道，设|ri-rp|=r，则r满足瑞利分布：

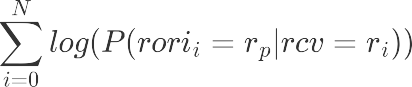
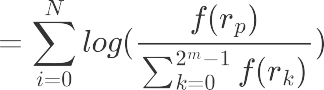




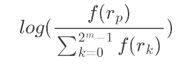
代入概率计算式则得出



回忆我们的优化目标——让每帧的概率取值乘积最大。现在它变成了下面这个问题：使

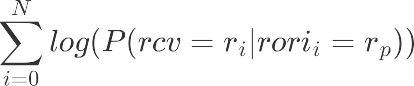
 

最大。则在viterbi的每个译码帧，可以基于计算下面形式的度量



完成“加比选”的操作，进而完成译码。

**基于最大似然的软判决**

假设在每帧，上式中的变化不大，则可以用最大似然替换最大后验来简化计算。此时我们的计算对象为

即

其中r的含义仍为|ri-rp|。只留下与r有关的项，则此时“加比选”的计算基础变为

**基于距离的软判决**

我们也可以简单地使

最小来近似地做viterbi译码。

上述三种方法中，区别仅仅在于viterbi译码加和的概率不同。为了统一接口，我们在解映射模块judging.m中直接将信道输出电平转化为对应的概率，供译码模块利用。关键代码如下：

后验概率计算：

1. dst = 2\*dst/(sigma^2).\*exp(-dst.^2/(sigma^2));
2. out = -log(dst/sum(dst));

其中dst为2m维向量，存储接收到的电平到各个标准复电平的距离，out为后验概率的对数度量。

似然概率计算：

1. out = -log(dst)+dst.^2/(sigma^2);

其中dst同上，out为似然概率的对数度量。

距离计算：

1. out = dst;

解释略。

由于所有的似然概率都建立在复电平间距离计算的基础上，因此一旦映射方式改变，卷积译码的方式也要做出相应的调整。例如，若采取2bit映射，则卷积译码网格中每列状态数为4；若采取3bit映射，则每列状态数为8。鉴于采取不同映射方式时信道性能也不同，因此，我们并不能准确地比较1/2卷积编码和1/3卷积编码的性能孰优孰劣。

## 三、CRC校验

CRC校验即在一段信息后填上校验位，使这段信息加上校验位后的整个序列可以整除校验子序列。设信息序列的多项式为c(x)，长n，校验子矩阵为g(x)，长m，则校验位r(x)可表示为

-xm-1c(x)mod(g(x))

在二元域中，某元素的加法逆元就是自身，因此负号可以去掉，只需要将-r(x)连接在在c(x)后即可使整个序列整除校验子。

序列除法的思路为：

* 将被除数和除数的最高位对齐并按位异或；
* 重复上一步，直到被除数幂次小于除数幂次。

序列除法代码如下：

1. pointer = 1;
2. n = length(dividend);
3. m = length(divisor);
4. while pointer <= n-m+1
5. if dividend(pointer) ~= 0
6. dividend(pointer:pointer+m-1) = xor(dividend(pointer:pointer+m-1),divisor);
7. end
8. pointer = pointer+1;
9. end
10. rem = dividend(end-m+2:end);

添加CRC校验位代码如下：

1. temp=[bitpatch,zeros(1,length(generator)-1)];
2. bitpatch = [bitpatch,poly\_rem(temp,generator)];

其中第一行bitpatch为待添加校验位的包，poly\_rem()为序列求余函数。

在CRC校验中，只需要检验每段比特流是否能被同一个校验子整除。需要注意的是，所有能够通过CRC校验的码字都可以写成d(x)g(x)的形式，由于添加校验位后的序列长n+m-1，g(x)最高次幂为m-1，因此d(x)最多有2n个，即能通过CRC的码字有2n个，而码字的总数为2n+m-1个；要想保证CRC校验的正确率，应当使能通过CRC的码字在总码字中的比例尽量小，因而m（校验子长度）应当尽量大。本次实验中，采取100011011作为校验子。

CRC校验的代码仅为对多项式除法函数的重复调用，为节省篇幅这里不再列出。

## 四、信道设计

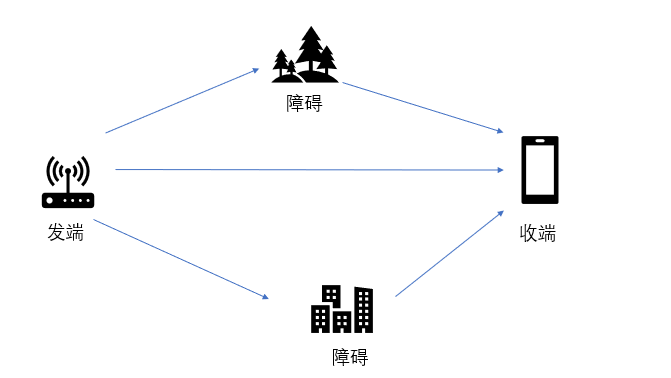
### 信道建模场景

本次实验采用的复电平信道中，输入和输出具有如下关系：

其中，

的实部虚部独立，服从。

以上建模描述了如下的场景：



在一个信道两侧，收端接收到的信号是发端发出的信号乘上信道增益，再加上一个高斯加性噪声。在我们的建模中，对应的就是信道增益，而对应噪声。

信道增益不仅包含发端到收端的直达增益，还包括无线信号经过障碍物后的多径增益，用表示。

由于收端往往是移动的，周围的环境情况经常变化，障碍物的特性差别很大。因此，多径增益并不是始终不变的（相应地，直达增益没有太大变化）。为简单起见，本实验的多径增益被建模为一个一阶自相关过程

基于以上的分析，任务中的三个传输场景也就很清楚了：

场景一：b=0，表示野外等几乎没有障碍物的环境。

场景二：b=ρ=1，表示障碍密集的城市中，收方保持静止的环境。

场景三：b=0.7，ρ=0.996：表示障碍较密集的城市中，收方持续移动的环境。

每次使用信道时，将要发送的电平拆分为份，即连续发送 次，最后输出电平并进行判决。另外，推导得部分信道特征的概率特性如下：



设则：

### 映射方式

对于复数电平映射，我们按照格雷映射的方式来进行。假设采用n比特/符号的电平映射，规定第个映射到复平面。

下面是1，2，3比特/符号的映射顺序：

1比特/符号： 0，1

2比特/符号： 00，01，11，10

3比特/符号： 000，001，011，010，110，111，101，100

### 三个场景的硬判决解映射讨论[[3]](#footnote-3)

根据不同的场景，我们设计了不同的硬判决方法。

**场景一：**

此时因此。

这时，借助上面的推导:且彼此独立。此时最大似然度量退化为欧氏距离度量，直接计算欧氏距离就可以实现判决。

**场景二：**

在这个场景下，信道增益完全由多径增益构成，因此相位增益完全不确定。为确定相位增益，发方可以和收方约定开始发送的一串比特的取值，根据收方收到的电平序列计算相位增益。由于多径增益在整个传输过程中不变，因此这个相位增益也是整个传输过程中的相位增益；据此，我们可以在之后接收的电平上消除这个相位增益再进行判决。

另外，根据概率论所学知识，此时。在此分布中，幅度增益更偏向于小于1，这个结论在后面的分析中会用到。

**场景三：**

在此场景中我们仍然可以通过估算相位增益的方式消除β变化对测量的影响。因此可以隔一段时间计算一次β，利用这些计算值对估计的相位增益进行更新。可以用移动指数平均方法，也可以用ARIMA等更复杂和准确的方法，按性能需要而定。

### 三个条件的讨论

**条件一：假设收发双方在完整一次通信过程之前，已提前知道所有的**

由于b是确定的，知道相当于知道，这意味着信道状态是完全可知的。

于是，在收方处可以将收到的码字乘上再进行硬判决。此时收到的码字叠加了一个的噪声，则此时噪声强度与成反比。要让噪声强度较小，则可以选择较大的时刻利用信道，这样既可以减少发送次数，节省能量，也能增大信噪比，提升传递准确性。

本次实验中，不对发送时刻进行选择，即每个时刻都发送。此时，若输入的电平符号为，连续调用了次复采样信道，输出的符号为，则：首先生成复采样信道的输入

然后送入复采样信道，之后输出。代入上式后推出

因此最后如果知道了，对于最后接收到的符号，需要乘上，再进行硬判决。

**条件二：假设收方提前知道所有的，而发方完全不知道**

此时信道状态对于收方单向透明，我们能做的只是选择模长较大的时刻进行解码，规避那些幅度增益较小的传输（具体原因和条件一相同）。

本次实验中不对发送时刻进行选择，故条件二的硬判决策略和条件一相同。

**条件三：假设收发双方均未提前知道**

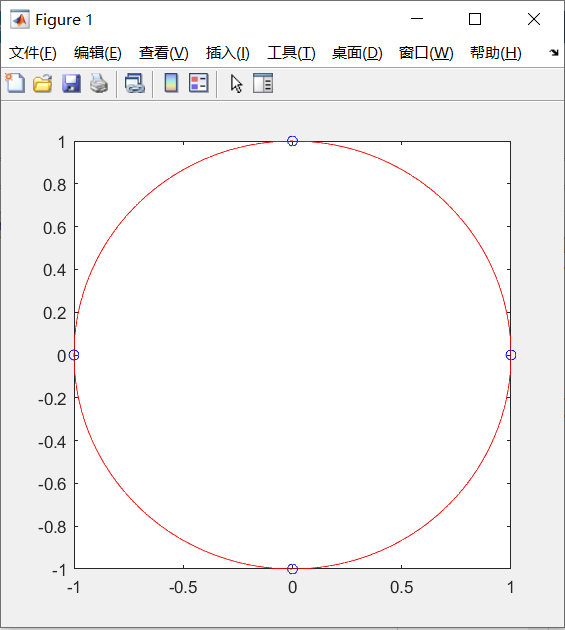
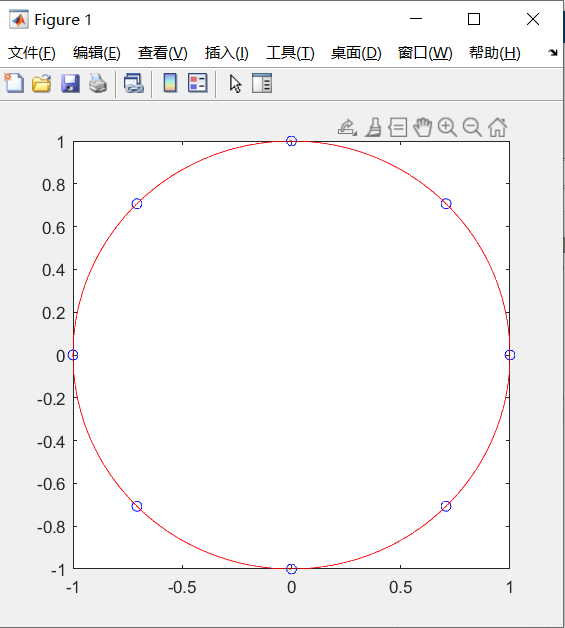
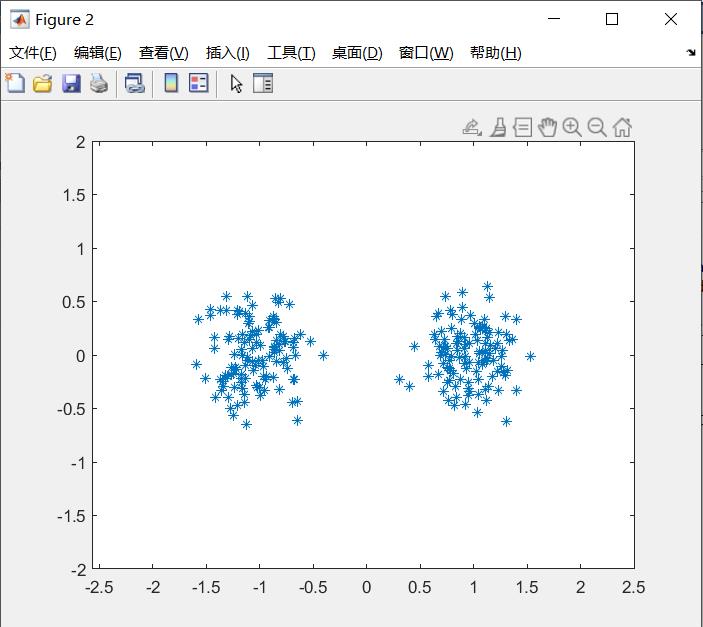
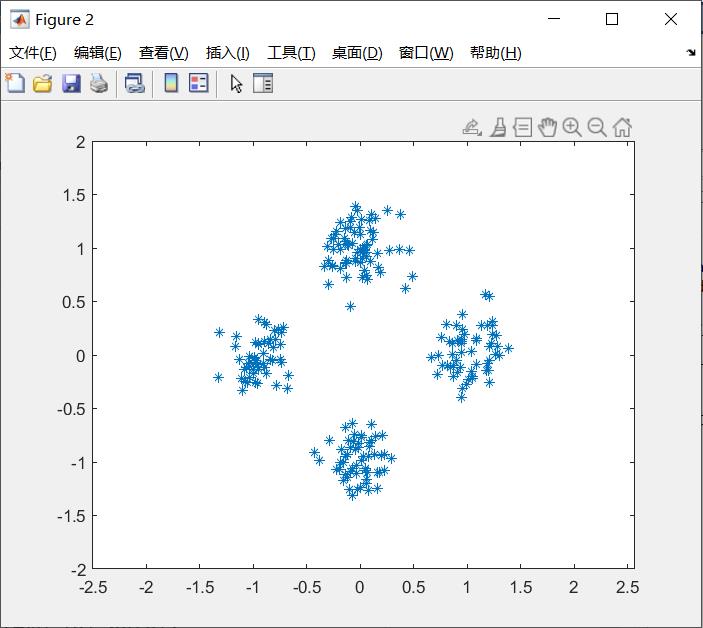
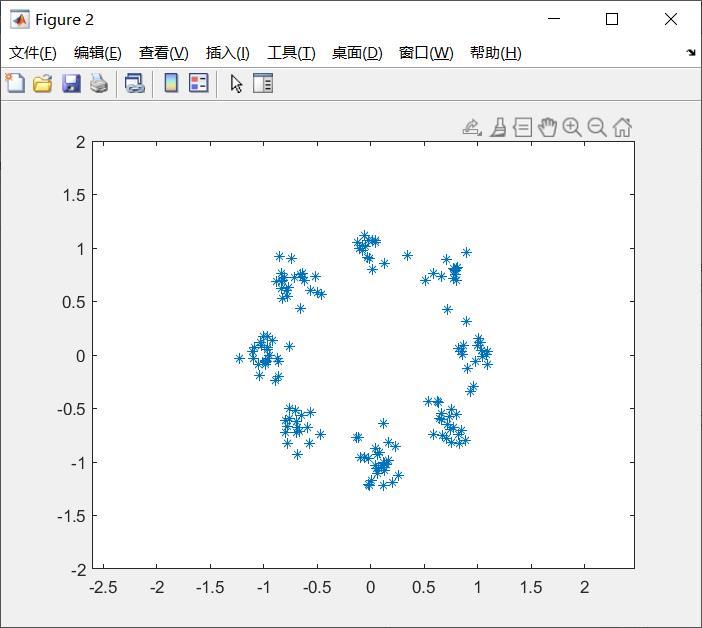
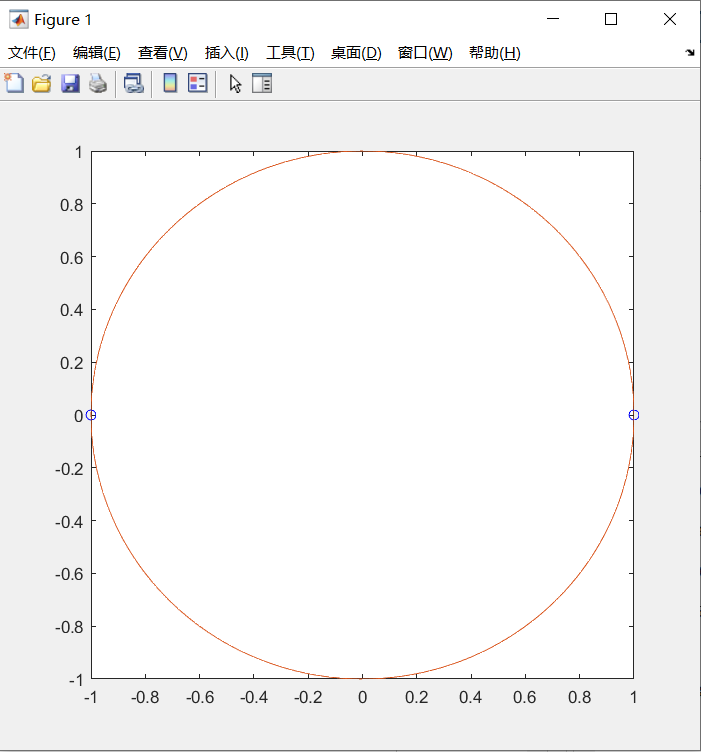
由前面的推导，此时：设则：

可以发现的均值趋向于和原始的图案在同一个直线上，而且按照我们的单位圆上的均匀编码图案，每个图案发送过去最后的方差都是一样的，因此最好的译码方式还是最短欧氏距离译码。在场景二和场景三的情况下，也可以对相位增益做出估计，在消除旋转的基础上再进行判决。

# 三、仿真结果及分析

## 信道部分

### 星座图

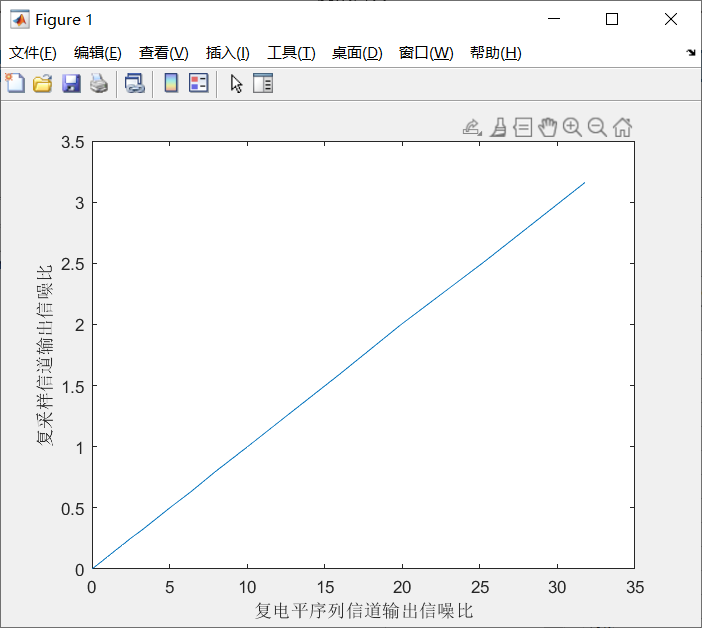
收发送方星座图见下：(1比特/符号，2比特/符号，3比特/符号)：

### 内核与外部信噪比关系

仿真前先给出外部信噪比的定义：

在完全没有噪声的情况下，收方接收到的电平为，将其定义为信号。在有噪声的环境下，收方接收到电平，故信噪比可以定义为：

反复使用信道测量信噪比，设置T=10，得到如下结果：



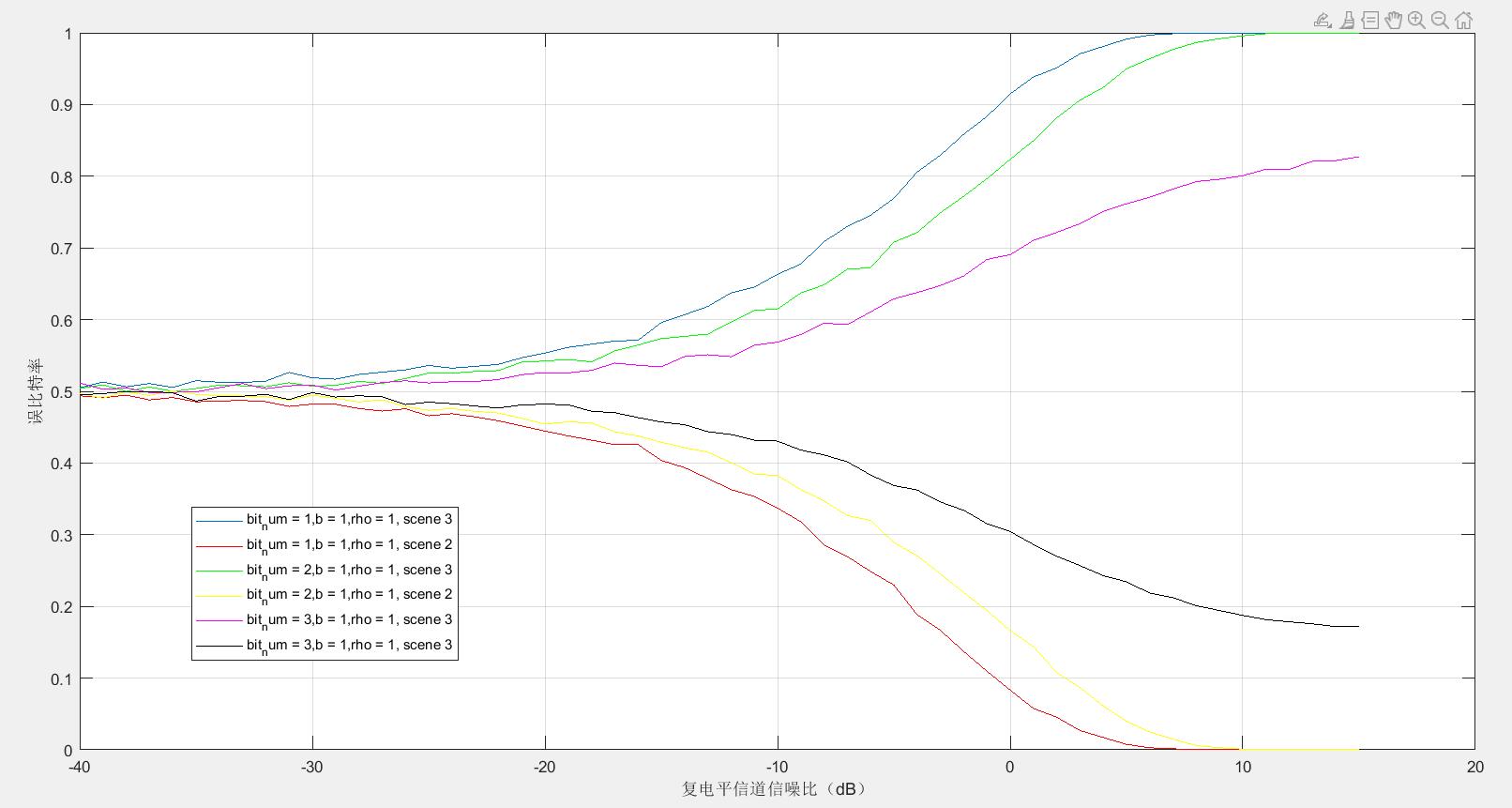
可以看出，外部信噪比恰为内部信噪比的10倍。

直观地说，我们将要发送的信号拆成10份，每一次发送都附带一次噪声，总共附带了十次噪声，因此信噪比就降低到一次使用的1/10。数学的证明利用期望的线性性质也容易给出。

### 硬判决误比特率与信噪比关系

**场景2**

设置条件2和条件3，，并采用1/2/3bit映射方式时，硬判决误比特率和信噪比的关系图线：

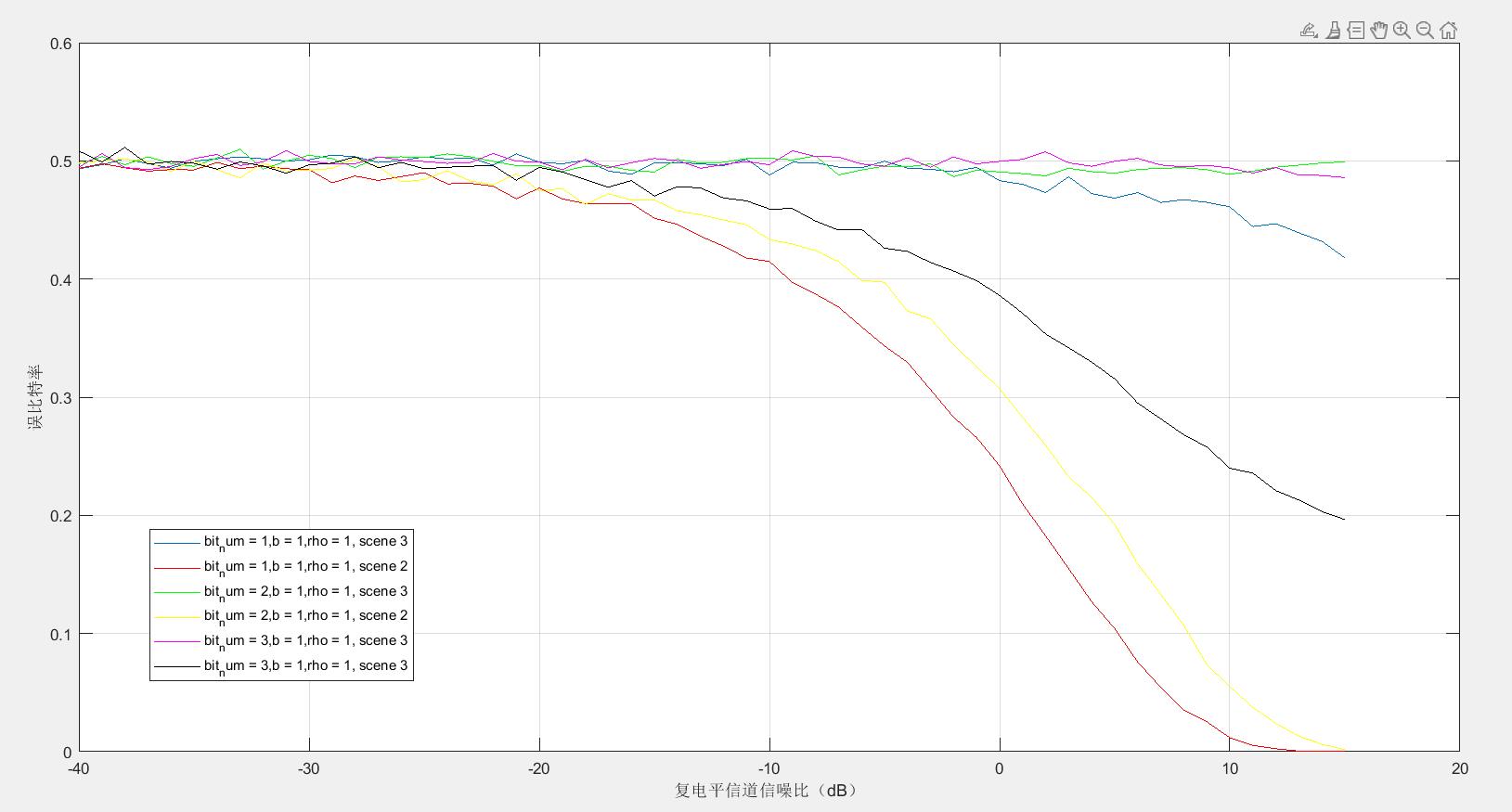


可以看到图线清晰地分成了向上和向下的两组。

首先分析向下的一组。组内的差别在于映射方式。可以看到，采用1bit映射的硬判决误码率要好于2bit，2bit好于3bit。原因是，我们在硬判决时使用了距离判决。对于相同的信噪比，标准电平在复平面上的距离越大，就越不容易产生误判。例如，在标准电平上附加一个模为0.5的噪声，若标准电平间距离为1，在收端就无法区别，但若距离为2则易于判决。我们把越多数据量映射到同一复电平，复平面上的点势必越密集，判别效果会越来越差。

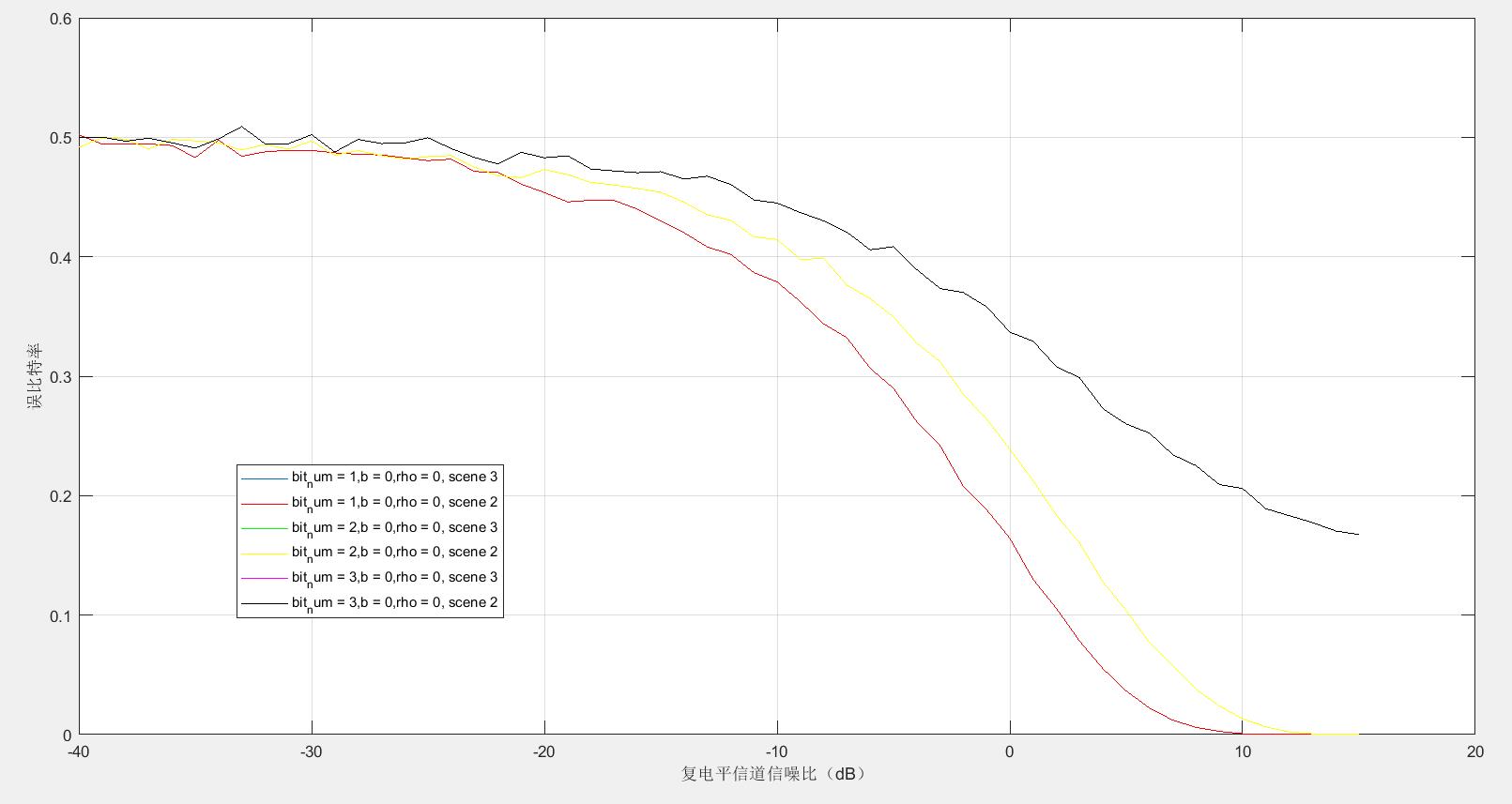
接着分析向上和向下的组间差异。对于情况二，需要预先估计β的值以消去相位增益。在某种程度上，“全错“也就等同于”全对“，只要把判决出的码字都进行反转就行了。向上的一组全错，显然是受到了β相位增益的干扰（恰巧反转了180°）。只要把它消除，就可以获得正确的码字（即向下的一组，收方事先知道β），这也说明了消除增益的过程是必要的。

改变β的辐角到π/2左右，得到如下的图：



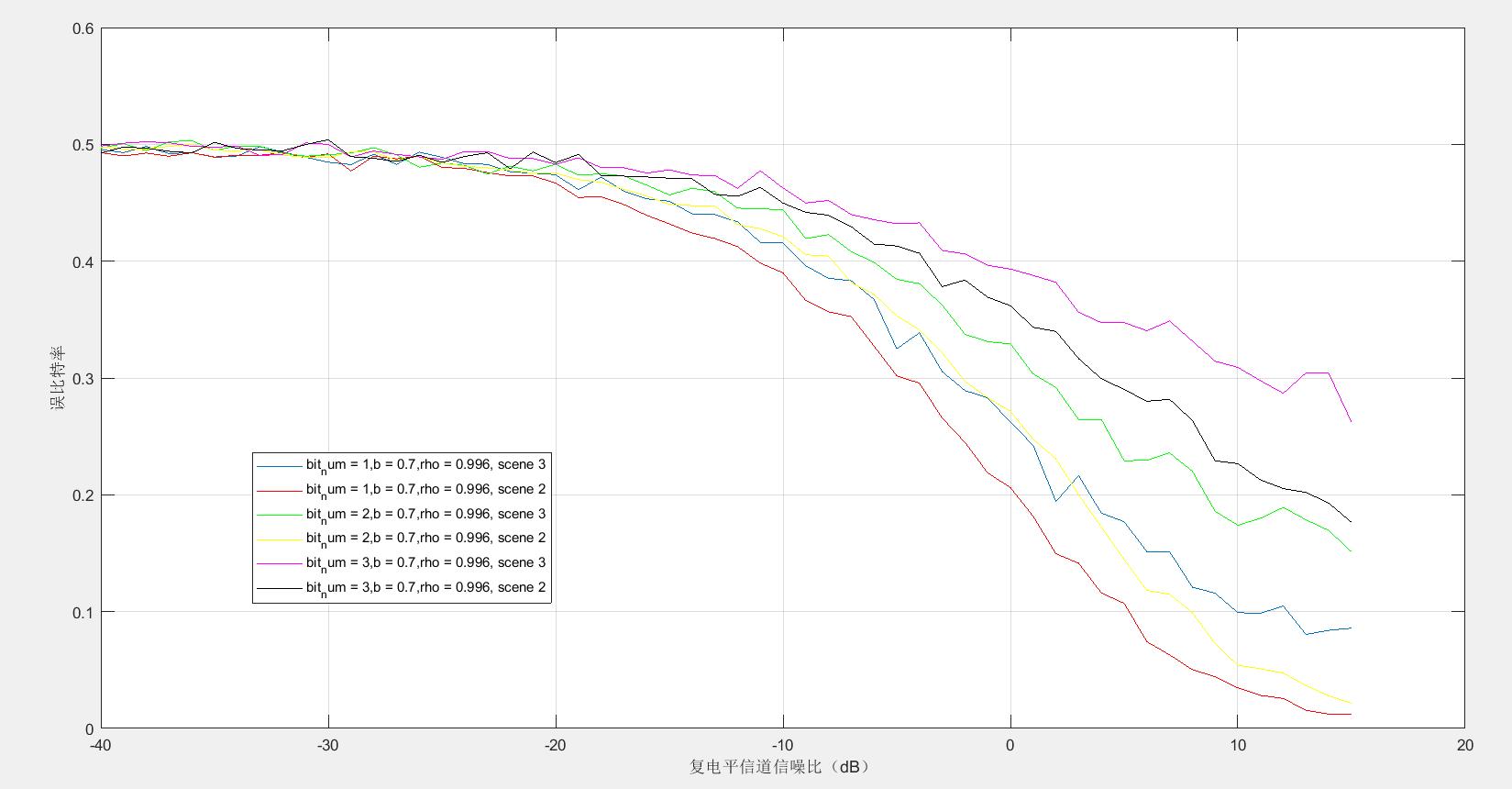
回忆标准复电平的分布。若复电平相位偏转了π，则所有bit都反转（汉明距离最大），而若偏转π/2，则只有部分bit反转。在这种情况下，不知道β的一组误码率情况不会持续增加（因为始终有bit没有反转）

**场景1**



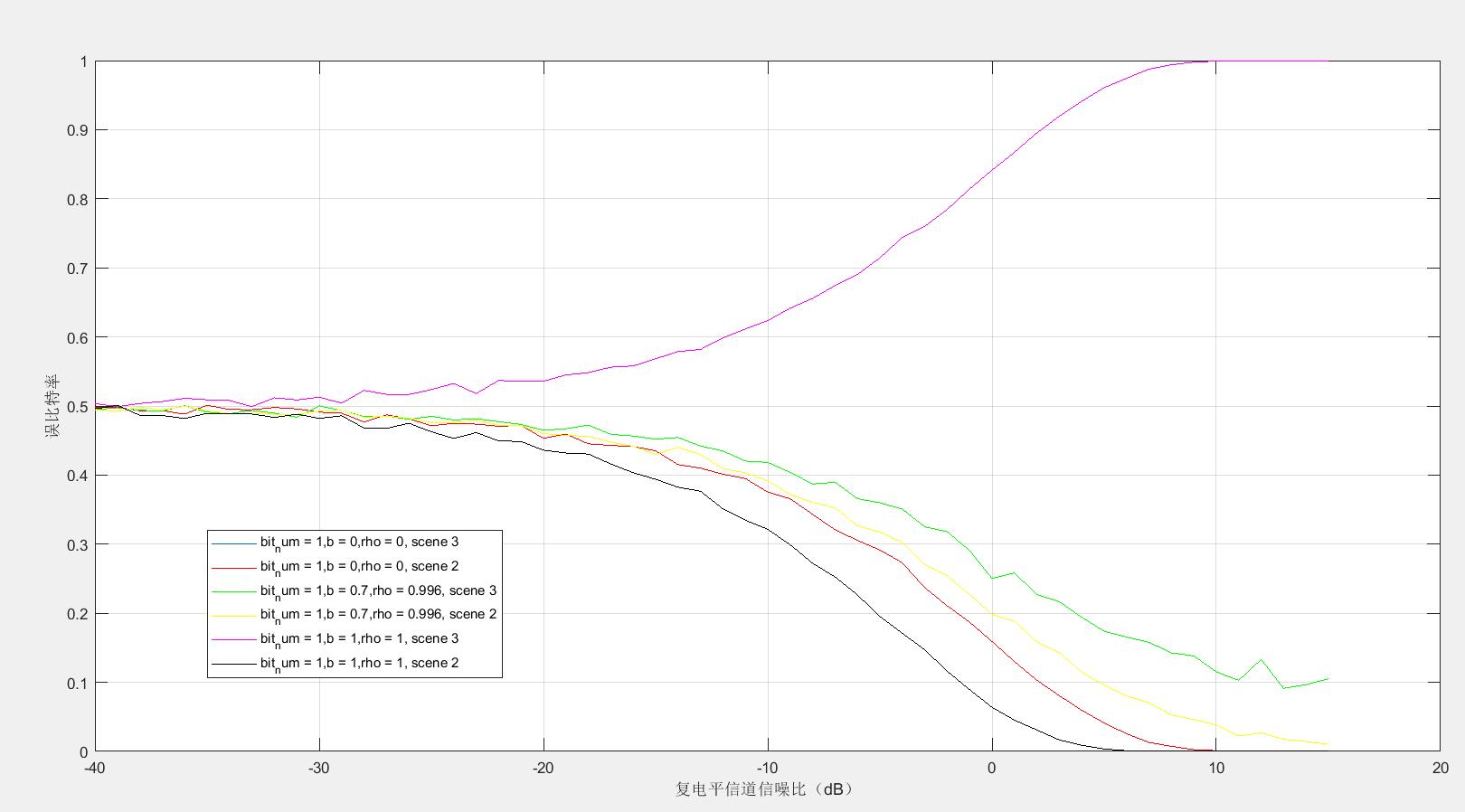
此时由于没有相位增益，知道β与否不会产生任何的区别。但是，采用不同bit映射的性能仍有差异，原因同上。

**场景3**



这个图案对应相位增益较小的情况。与上面两种情况不同的是，每条曲线的性能都存在波动，这对应着β变化引入的随机性。

**相同比特数映射，不同场景：（以1bit为例）**



图中图线从上到下依次为：

场景2，条件3

场景3，条件3

场景3，条件2

场景1，条件2/3

场景2，条件2

由条件3下存在上扬曲线可以得知，本次β的辐角应该在π左右。从上图中我们可以得出如下结论：

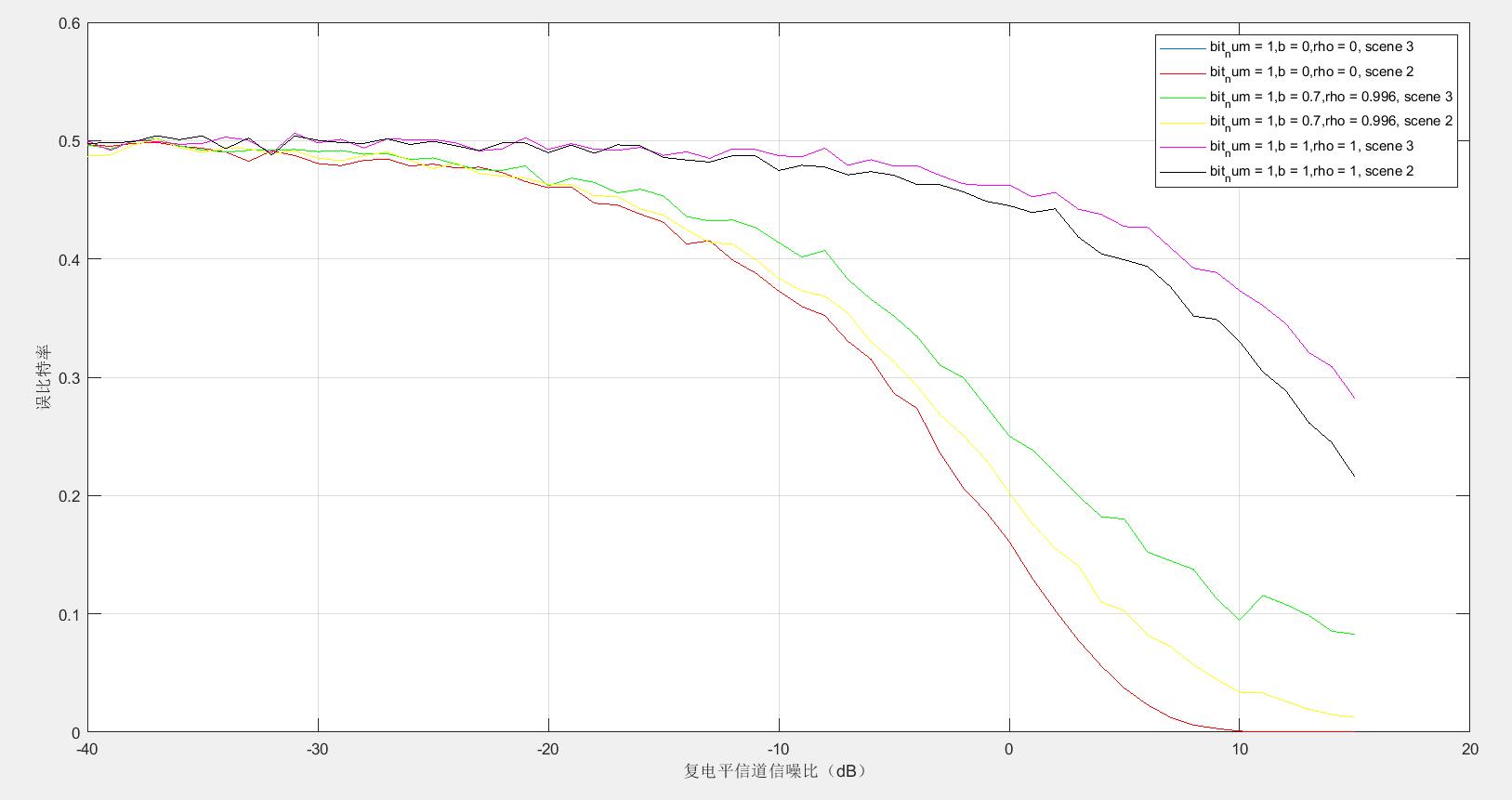
（1）场景三会引入性能的波动。

（2）由于β的变化，场景三中信道性能最差。

（3）对于部分β，场景2性能优于场景1。

对于第三点，我们可以看到多径带来的幅度增益提升了信道性能，使得其优于没有多径的情况。实际上，早已存在RAKE等技术利用这一现象，用多径增益提高信道性能。

（作为对比，我们设置|β|<1，可以发现场景2的性能较场景1差）

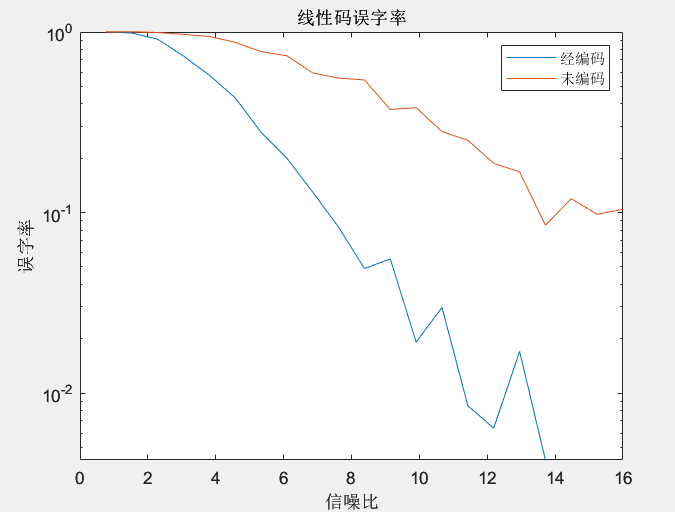


## 编译码部分

### 线性码

#### 线性码的性能评价

在b=0,ρ=0情况下调节信噪比并计算线性编码的误字率，如下：

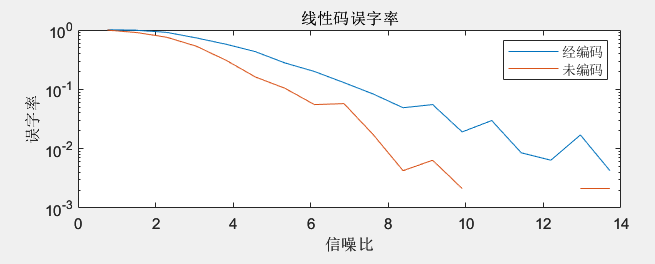


从上图可以得出如下结论：

（1）误字率随信噪比增大而下降，当信噪比足够大时，线性分组码可以消除所有误字；

（2）在同一信噪比、同一功率下，编码后的误字率远低于未编码的误字率。

进一步地，考察线性码的能量性能。将线性码误字率与相等能量未编码的误字率对比，如下：

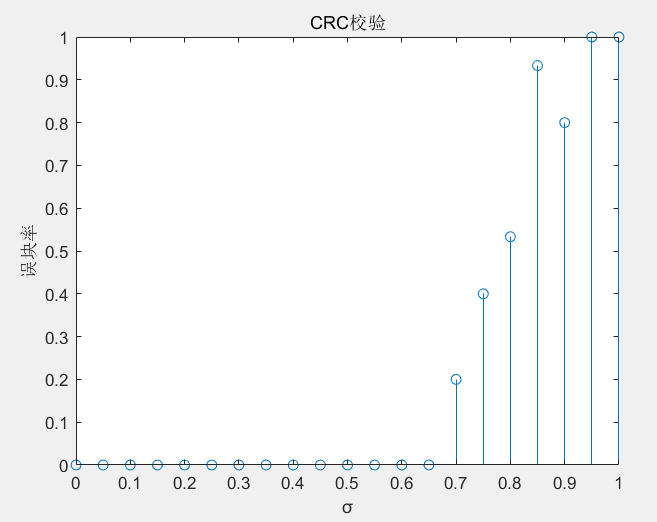


横坐标中的信噪比为线性码使用信道的信噪比，而“未编码”的误字率是在信噪比为原先3/8的信道中测量的。可以看到，由于我们使用的（8，3，4）码性质不佳，在同等能量下相较于未编码、只经过解映射的传输方式，误字率不降反升，不仅如此，还增加了译码时间和使用信道的次数。唯一的优点是降低了功率，对放大器等硬件的要求较小。

### 卷积码

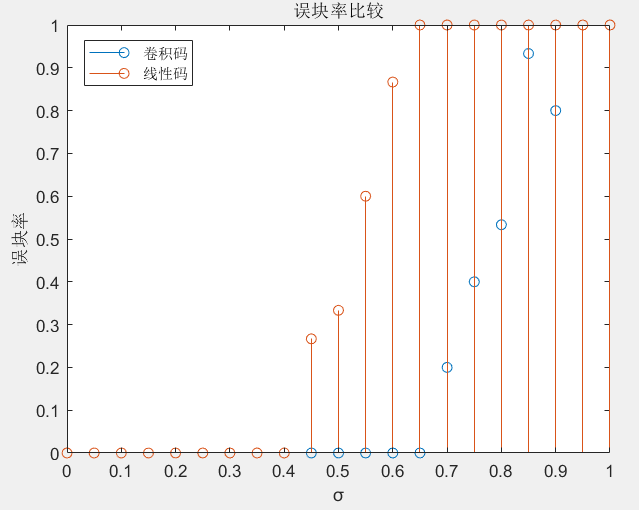
#### 用CRC分析卷积码误包率

将长度为3000的码流经1/2效率卷积编码后再发送，通过b=0,ρ=0的信道，卷积编码采用软判决（最大似然），映射方式为2bit映射。用CRC检测误包率，得到误包率和σ的关系如下：



信道的信噪比为2/σ2。σ未达临界值时，信道噪声造成的错误可以被卷积码完全纠正。当σ达到临界值以上，卷积码不足以纠错，错误开始在码字各处出现并污染码块，误块率大幅增长并使得大部分的块都出错。

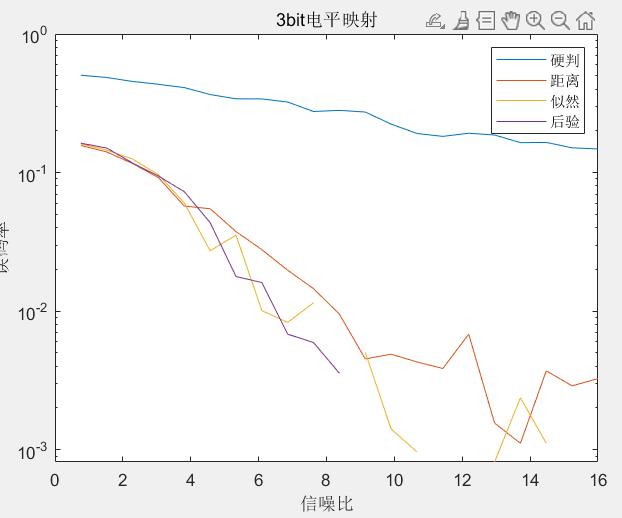
卷积码在保证误块率上有特殊的优势。在同一条件下，将其和线性码的误块率比较，得到如下结果：



容易看到，在保证块正确性上，卷积码的性能远优于线性分组码。这是因为，卷积码有将同一段原码映射到不同时刻编码上的能力，即使某段信息出错，也能通过附近位置的编码恢复。另外，相同信噪比条件下，卷积码（效率1/2）的性能优于线性码（效率3/8），这也证明卷积码在相同能量下能保证更大的正确率。当然，这是以编/译码的时空和结构复杂度的提升为代价的。

#### 分析卷积码在不同度量下的性能

采取3bit映射，b=0，ρ=0，在不同信噪比下测定不同卷积译码方式的误码率，得到如下结果：



其中未能画出的点表示误码率为0（纵轴采用对数坐标故无法画出），图上的波动主要由β1选择的随机性导致。从中可以得出如下的结论：

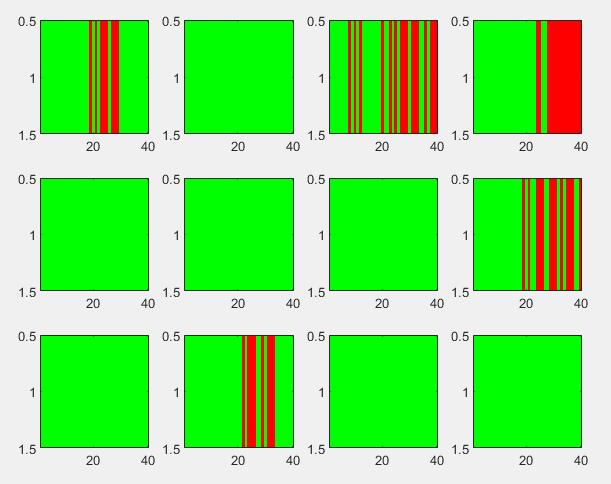
（1）无论软判采取何种判决标准，性能都远优于硬判；

（2）总体来说，性能上最大后验>最大似然>距离度量，但区别主要集中在高信噪比区域。随着信噪比增加，三者趋于一致，当信噪比<6dB时已经无法通过改变判决方式获得明显的性能提升；

（3）考虑到最大似然度量与最大后验度量效果区别不大，为节省计算复杂度，采取最大似然度量有时比较合适。

#### 分析卷积码的错误图案

在b=0.7,ρ=0.996,σ=0.05条件下，将一小段长为40bit的信息进行1/3效率卷积编码，随机选取不同的β1进行12次传输并绘制对应的错误图案，得到如下结果：



从上图中可以看到卷积码的记忆特性。在同样的信噪比下，卷积码有机会纠正全部的错误并译出正确的码字；但也有可能从某一点开始出错，并由于记忆性影响之后的译码过程，导致错误聚集出现。当然，在聚集的错误中也可能译出少量正确的码字，但总体上错误仍然是成块出现的。上图中，全部正确的译码结果占一半以上，在出现错误的结果中，错误图案总是集中出现并占据相当的长度，不会出现单独1位的错误。

# 四、特定传输任务中编译码及映射方式的选择

**要求：**

某传输任务信息量为1200bit，T=5,允许使用复电平采样信道次数分别为4/5/6/7.5/9千次时，给出合理的编码、映射、解映射、译码设计。

**分析：**

对应的复电平信道使用次数分别为1800/1500/1200/1000/800(从高到低)次。我们将这些条件分为两组分析：

1、1800/1500/1200

首先考虑卷积编码。软判决情况下，在我们的设计中，卷积编码效率和电平映射效率是耦合的，互为倒数。因此，如果选择卷积编码方式，则复电平信道的使用次数和需要传输的比特数是恰好相同的，这里是1200次。在这一组信道使用条件里，足够传输1200次，因此可以用卷积编码。在实验中发现，同等信噪比下2bit映射表现较好，故使用1/2效率卷积编码和2bit映射。

接着考虑线性编码。经过前面的分析，在同等能量（即同等信道使用次数）下，本次实验中的线性分组码性能总劣于软卷积码，但优于硬判决卷积码。因此，除非设计时有硬件复杂度和译码时间等要求，在考虑传输准确性时总应该选择软判决卷积码作为编码方式。

剩下的问题是，在1800/1500次使用条件下，多余的使用次数如何利用？此时可以使用线性码，调整编码方式中n和k的大小，增加冗余来提高纠错能力。由于此时映射和编码已经解耦，可以进一步提高映射的比特数以获得更高的冗余度和纠错能力，只是要付出复电平抗干扰能力降低的代价。

2、1000/800

此时信道使用次数少于1200，已经不能使用软判决卷积码。采用3bit映射，可以让能够发送的码字增加到3000/2400，从而可以编写5/2或2效率的线性码。为尽可能提高纠错能力，可以采取BCH等编码技术。若采用2bit映射，则效率为5/3或4/3，冗余太少，不足以保证纠错能力。

# 五、实验心得

**邵晨扬：**

编码模块在掌握原理之后写起来其实很简单，但是译码的viterbi在实现起来还是有些难度的。以为在课上听懂了算法原理，但写起来很多细节都不是很好写。就像那个如何通过状态值反推前置的两个状态值以及路径信息，有点费脑筋。但最后把处理的思路理清楚之后，代码的难度就不高了。还有一个很值得一提的就是软判算法的研究，我本来以为得到一个实数序列是很简单的事情，后来发现只是对于1bit/符号的映射比价简答，对于2bit/符号以及3bit/符号，还是很困难的。但最后还是提出了一种解决方案，并且效果不错，还是很有成绩感和收获的。

**唐钰凯：**

一开始实现信道让我感觉很没有信心，因为太多随机信号加在信道上，让我觉得很难检验信道程序编写的正确性。但是慢慢的我找到了方法，应该一部分一部分分别编写，而且应该先从简单的情况开始编写，例如，这样随机性就会减弱一点，也就更好检查自己程序的正确性。另外就是我写的信道在整个程序中是提供接口的关键位置，因此下次要减少变动，早点确定好，以便我的队友们调用。另外在画图过程中也是发现了很多之前没有发现的有趣现象，也值得我们的思考和探索。

**张鹤龄：**

本次我负责的内容代码难度并不是很高，主要难度集中在数学推导方面。

首先，线性码解码中，需要用得到的校正子查表，得到对应的错误图案。表的顺序有两种选择：一种是根据错误图案重量从轻到重排序，第二种是根据校正子序列对应的整数排序。经过数学推导，发现第一种方案近似为多项式复杂度，而第二种由于校正子和整数一一对应，只需要常数复杂度，因此最后的编码中选择了第二种方案。

其次，Viterbi译码的度量推导也有些难度。虽然计算上不是很复杂，但尝试推导这个度量也让我对Viterbi译码的数学基础有了更深刻的认识。同时，基于似然度量和后验度量的译码确实相当程度上提高了译码准确性，这一点顺利得超出预期。

除此之外，通过这次实验我们也积累了一些合作项目的经验，比如git和github的使用方法、接口的编写和代码规范等等。相信这些经验会在今后的实验中发挥作用。

# 附录 主要文件清单及说明

bsc\_channel.m:

比特映射为电平，加上信道传输

judging.m:

提供硬判决或软判决的输出

plot\_all.m:

信噪比与误码率关系曲线的各种对比。

plot\_all\_2nd.m:

信噪比与误码率关系曲线的各种对比。

plot\_all\_3nd.m

信噪比与误码率关系曲线的各种对比。

plot\_ber\_snr\_curve.m

画内外信噪比关系，信噪比与误码率关系曲线

main\_nego.m

画信道的典型误码图案与辐角对比图。

Convol\_Code.m:

卷积码编码模块所在文件。

Convol\_Decode.m:

卷积码译码模块所在文件。

Convol\_DecodePro.m:

卷积码软判译码模块（特别解决2bit/符号和3bit/符号的情况）。

ConvolMain.m:

卷积码编译码整体调试文件。

GenerateSuiji.m

卷积码编译码单方面调试的时候，人为将01序列转换为实数序列。

CRC\_generator.m

为比特流添加CRC校验位。

CRC\_checker.m

通过CRC检测是否有误包。

linear\_coder.m

线性分组编码模块。

linear\_decoder.m

线性分组编码模块。

poly\_rem.m

多项式求余模块。

main\_AllergoP.m

绘制CRC误包率、线性码误字率模块。

1. 由于我们已经得到了符合预期的仿真结果，且对各部分模块进行了单独调试和验证，因此可以保证模块的正确性，这里不再赘述单独模块的验证过程。若读者有兴趣验证，可以访问我们的[github仓库](https://github.com/AllergoP/IntroductionOfCoding/tree/main/Task1_Group_And_Conv_Code)。 [↑](#footnote-ref-1)
2. 这里左乘和右乘只差一个转置，因此不必特意区分。 [↑](#footnote-ref-2)
3. 软判决法在2-3bit映射情况下，不输出确定的判决结果，只输出接收电平到各个标准电平的距离及对应的度量，供卷积译码模块使用。在1bit映射的情况下，接对输出的实部取tanh，就可以得出一个有界的输出。 [↑](#footnote-ref-3)